Segurança e Disponibilidade através de Resiliência Proactiva

Paulo Sousa

DI–FCUL TR–08–15

Junho 2008

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Segurança e Disponibilidade através de Resiliência Proactiva

Paulo Sousa
LaSIGE - Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa

Prémio Científico IBM 2007

Resumo

Desde o aparecimento da Internet, a segurança dos sistemas informáticos é uma preocupação crescente da sociedade. Por um lado, as ferramentas de ataque a sistemas informáticos estão cada vez mais acessíveis e os atacantes têm vindo a aumentar a eficácia e furtividade dos seus ataques. Por outro lado, os piratas informáticos têm começado a desviar a sua atenção para as infra-estruturas críticas (por exemplo, redes de distribuição de electricidade, água, gás) uma vez que, por razões de eficiência operacional, as mesmas usam redes públicas como a Internet para interligarem as suas várias instalações (por exemplo, sub-estações de energia eléctrica espalhadas por um país são monitorizadas e comandadas por centros de controlo regionais). Neste cenário, torna-se imperativo dotar os sistemas informáticos críticos de mecanismos que lhes permitam ser resilientes mesmo na presença de ataques bastante severos que podem ser tipicamente difíceis de detectar em tempo útil. Para além disso, esta resiliência deve ser assegurada de forma automática durante todo o tempo de vida do sistema informático.

A primeira parte do trabalho introduz um novo predicado de segurança, *segurança-contra-exaustão*, e estuda as condições necessárias e suficientes para se construir um sistema informático seguro contra a exaustão de nós. Note-se que um sistema informático é tipicamente constituído por bastantes nós, isto é, um sistema informático é tipicamente distribuído. Alguns destes nós podem servir como backup ou permitir o mascaramento de erros (tolerância a faltas) e existe tipicamente um número mínimo de nós correctos N_m necessário ao correcto funcionamento do sistema distribuído. Diz-se que um sistema distribuído é seguro contra a exaustão de nós quando garante, durante todo o seu tempo de vida, que existe sempre um número de nós correctos maior ou igual do que N_m.

No contexto específico da tolerância a faltas, o trabalho prova que é impossível, sob o modelo assíncrono (isto é, quando não são feitos quaisquer pressupostos temporais), construir-se um sistema distribuído tolerante a faltas (acidentais ou maliciosas) *seguro contra a exaustão de nós*. A importância deste resultado advém da importância que a tolerância a faltas tem vindo a ganhar nos últimos anos, nomeadamente na vertente da tolerância a faltas maliciosas, e por se acreditar que os sistemas mais seguros seriam aqueles onde não são feitos pressupostos temporais. Ora, este trabalho mostra precisamente que sem pressupostos temporais não é possível garantir-se segurança-contra-exaustão de nós.

A segunda parte do trabalho apresenta um novo paradigma, *resiliência proactiva*, para a construção de sistemas distribuídos tolerantes a faltas seguros contra a exaustão de nós. A resiliência proactiva é baseada em hibridização da arquitectura e modelação híbrida de sistemas distribuídos: o sistema é maioritariamente assíncrono e faz uso de um subsistema síncrono para *recuperar periodicamente* os nós e remover os efeitos das faltas/ataques.

Por fim, o trabalho descreve um cenário de aplicação da resiliência proactiva. Neste contexto, é apresentada um nova arquitectura para a *replicação de máquina de estados* tolerante a faltas. É estabelecido um novo resultado de que um mínimo de $3f + 2k + 1$ réplicas são necessárias para garantir resiliência e disponibilidade, num sistema onde f faltas arbitrárias possam acontecer entre recuperações, com um máximo de k réplicas a recuperar ao mesmo tempo.

Palavras-Chave: Sistemas distribuídos, confiabilidade, resiliência, disponibilidade.

Classificação ACM:
D.4.5 [Operating Systems]: Reliability — Fault-tolerance;
D.4.7 [Operating Systems]: Organization and Design — Distributed systems;
C.2.0 [Computer-Communication Networks]: General — Security and Protection

1 Introdução

Hoje em dia, e cada vez mais, a confiabilidade dos sistemas informáticos é um assunto importante, dado que os computadores estão a invadir as nossas vidas, criando uma dependência cada vez maior no seu correcto funcionamento.

Informalmente, diz-se que um sistema é *confiável* se existe uma probabilidade alta de se comportar de acordo com a sua especificação. Quando o comportamento de um sistema viola a sua especificação, dizemos que ocorre uma *falha*. Construir sistemas confiáveis é construir sistemas onde se evita que as falhas aconteçam. Para se ter sucesso, é necessário perceber primeiro o processo que leva à falha, e que tipicamente é desencadeado por uma causa interna ou externa, denominada *falta*. As faltas são assim os alvos naturais de muitos dos mecanismos existentes para se obter confiabilidade, tais como: previsão de faltas, remoção de faltas, prevenção de faltas.

Claro que não é possível garantir que nenhuma falta vá ocorrer durante a operação do sistema. Logo, é necessário criar mecanismos complementares que bloqueiem o efeito da falta antes desta gerar uma falha. Quando este tipo de mecanismos está presente, diz-se que o sistema é capaz de oferecer um serviço correcto apesar da ocorrência de uma ou mais faltas, ou, em outras palavras, diz-se que o sistema é *tolerante a faltas*.

Durante a concepção de um sistema tolerante a faltas, são feitos pressupostos sobre o ambiente em que o sistema irá executar. Nomeadamente, o arquitecto de sistema faz pressupostos sobre o tipo e número de faltas que podem acontecer e sobre o comportamento temporal dos vários componentes do sistema. Neste contexto, um objectivo fundamental para se ter confiabilidade é garantir-se que durante a execução do sistema, o número *real* de faltas nunca exceda o número máximo de faltas f que se decide tolerar em tempo de projecto. Em termos práticos, o arquitecto de sistema deveria prever o número máximo de faltas N_f possíveis de acontecer durante a execução do sistema, de forma a que o mesmo fosse concebido para tolerar pelo menos $f \geq N_f$ faltas. O presente trabalho mostra que a dificuldade de se atingir este objectivo varia não só com o tipo de faltas considerado, mas também com os pressupostos temporais. Para além disso, os modelos de sistema usados actualmente escondem parte destas dificuldades, uma vez que não são suficientemente expressivos. O trabalho propõe um novo modelo teórico de sistemas (REX), suficientemente expressivo para representar esses problemas. O modelo REX introduz uma nova dimensão segundo a qual a confiabilidade dos sistemas pode ser avaliada, *segurança-contra-exaustão*. Segurança-contra-exaustão significa segurança contra exaustão de recursos e o seu significado concreto no contexto de um determinado sistema depende do tipo de recurso que se considere. O trabalho foca os nós de um sistema distribuído tolerante a faltas e estuda as condições em que o pressuposto típico sobre o número máximo de falhas de nós pode ou não ser

violado. Um sistema distribuído tolerante a faltas *seguro contra a exaustão de nós* é um sistema que garantidamente não sofre mais do que o número máximo assumido de falhas de nós.

Um primeiro resultado interessante deste trabalho é a conclusão de que é impossível, sob o modelo assíncrono (isto é, não fazendo qualquer pressuposto temporal sobre o tempo de processamento local e o tempo de entrega de mensagens pela rede), construir-se um sistema distribuído tolerante a faltas seguro contra a exaustão de nós. A importância deste resultado advém da importância que a tolerância a faltas tem vindo a ganhar nos últimos anos, nomeadamente na vertente da tolerância a faltas maliciosas, e por se acreditar que os sistemas mais seguros seriam aqueles onde não são feitos pressupostos temporais. Ora, este trabalho mostra precisamente que sem pressupostos temporais não é possível garantir-se segurança-contra-exaustão de nós.

O resultado de impossibilidade motivou a investigação no desenvolvimento do modelo e arquitectura adequados para garantir segurança-contra-exaustão de nós. No contexto desta investigação, percebeu-se que trabalhos anteriores já tinham proposto uma abordagem, denominada *recuperação proactiva*, com objectivos similares. A recuperação proactiva, que pode ser vista como uma forma de redundância dinâmica, tem o potencial de permitir a construção de sistemas tolerantes a faltas (acidentais e maliciosas) seguros contra a exaustão de nós. O objectivo da recuperação proactiva é conceptualmente simples: os nós são rejuvenescidos periodicamente para que sejam removidos os efeitos de faltas/ataques que tenham entretanto ocorrido. Se os rejuvenescimentos ocorrerem a um ritmo adequado, então um adversário é incapaz de corromper nós suficientes (isto é, mais do que aqueles que o arquitecto do sistema assumiu poderem ser corrompidos) para comprometer o sistema distribuído. No entanto, para dar estas garantias, a recuperação proactiva precisa de ser concebida segundo um modelo suficientemente forte que lhe permita atingir o seu objectivo: o rejuvenescimento regular do sistema. Nenhum dos trabalhos existentes consegue garantir este objectivo. De facto, os problemas particulares dos diferentes trabalhos existentes podem ser categorizados em quatro classes:

1. Um adversário pode ser mais forte (ter mais potência) do que o originalmente assumido e corromper os nós a um ritmo mais rápido do que as recuperações.

2. Um adversário pode tentar atrasar o ritmo das recuperações, de forma a aumentar as hipóteses de comprometer o sistema com a potência disponível.

3. Um adversário pode fazer ataques furtivos às referências temporais do sistema, o que em sistemas assíncronos ou parcialmente síncronos pode até nem ser detectado pela lógica essencialmente atemporal do sistema, deixando-o indefeso.

4. Os procedimentos de recuperação podem fazer com que os nós fiquem num estado temporariamente inactivo, baixando o quorum de redundância e a resiliência do sistema.

O primeiro problema, violação dos pressupostos sobre a potência do adversário, está fora do âmbito do trabalho, e é um problema irresolúvel, uma vez que envolve pressupostos fundamentais na área de investigação da tolerância a faltas. Este problema pode no entanto ser combatido com técnicas que mitiguem as possibilidades de o adversário ganhar vantagem de uma forma não prevista, tais como, diversidade, mutação, ofuscação, ou componentes seguros. Todos os restantes problemas são resolvidos pela abordagem proposta neste trabalho, *resiliência proactiva*, um novo paradigma para a construção de sistemas distribuídos tolerantes a faltas, que permite usufruir de todas as potencialidades da recuperação proactiva. A resiliência proactiva é baseada num modelo e arquitectura híbridos: o sistema (distribuído) é maioritariamente assíncrono e faz uso de um subsistema síncrono para recuperar periodicamente os nós e remover os efeitos das faltas/ataques. O trabalho descreve o modelo genérico de resiliência proactiva (PRM), que por sua vez modela o subsistema de recuperação como um componente abstracto denominado PRW. O componente

PRW pode ter várias instâncias, dependendo dos requisitos do protocolo de recuperação que for adequado em cada caso (por exemplo, rejuvenescimento de chaves criptográficas, restauro do código do sistema operativo e/ou das aplicações).

Por fim, o trabalho apresenta um cenário de aplicação da resiliência proactiva. É descrito como é que a resiliência proactiva pode ser usada para aumentar a resiliência e a disponibilidade de uma máquina de estados replicada tolerante a faltas. Neste contexto, é apresentada um nova arquitectura (baseada no PRM) para máquinas de estado replicadas tolerantes a faltas arbitrárias. Esta arquitectura usa uma instância do componente PRW para remover periodicamente os efeitos das faltas/ataques das réplicas. É feito um estudo quantitativo do nível de redundância necessário para se conseguir ter replicação de máquina de estados resiliente e disponível, e neste contexto é estabelecido um novo resultado: um mínimo de $3f + 2k + 1$ réplicas são necessárias para tolerar f faltas arbitrárias entre recuperações, com um máximo de k réplicas a recuperar ao mesmo tempo.

Principais contribuições deste trabalho:

1. Um predicado de segurança, segurança-contra-exaustão, e o estudo das condições necessárias e suficientes para se construir um sistema informático seguro contra a exaustão de nós. Neste contexto, é apresentado um resultado de impossibilidade: é impossível a construção de sistemas seguros contra a exaustão sob o modelo assíncrono (isto é, não fazendo pressupostos temporais).

2. Um paradigma, resiliência proactiva, para a construção de sistemas distribuídos tolerantes a faltas e seguros contra a exaustão de nós. O paradigma assenta num modelo e arquitectura híbridos, oferecendo um bom compromisso em termos de pressupostos temporais, permitindo concentrar os pressupostos temporais necessários à segurança-contra-exaustão num subsistema de recuperação mais protegido. O sistema propriamente dito (isto é, a parte que é recuperada) pode ser projectado sob o modelo assíncrono, mantendo-se assim imune a faltas de qualquer tipo.

3. Um cenário de aplicação da resiliência proactiva, que demonstra a aplicabilidade do paradigma teórico num cenário prático. Para além disso, é estabelecido um novo resultado sobre o número mínimo de réplicas necessário para se garantir a resiliência e disponibilidade de uma máquina de estados replicada.

Todas as contribuições acima referidas fazem parte do trabalho de doutoramento do autor deste documento e são portanto fundamentalmente da sua responsabilidade. Versões preliminares de algumas das contribuições foram publicadas em actas de conferências internacionais, mas o trabalho tal como é descrito neste documento ainda não se encontra publicado, estando submetido para publicação em duas revistas internacionais. Adicionalmente, o paradigma da resiliência proactiva está actualmente a ser usado na construção de dispositivos de protecção para infra-estruturas críticas do sector eléctrico, no contexto de um projecto europeu em que o autor deste documento participa.

2 Segurança-Contra-Exaustão

Tipicamente, a correcção de um protocolo depende de um conjunto de pressupostos tendo em conta aspectos como o tipo e número de faltas que podem acontecer, a sincronia da execução, etc. Estes pressupostos são na realidade uma abstracção dos recursos reais que o protocolo necessita para funcionar correctamente (por exemplo, quando um protocolo assume que as mensagens são entregues num determinado prazo, está de facto a assumir que a rede vai ter certas características

de largura-de-banda e latência). A violação destes pressupostos sobre recursos pode afectar a correcção e/ou progresso do protocolo. Se o protocolo é vital para a operação de algum sistema, então a própria correcção e/ou progresso do sistema podem também ser afectado.

Para se poder estudar a segurança-contra-exaustão de sistemas em relação a determinados pressupostos sobre recursos, é necessário adoptar um modelo conveniente. Seja φ_r um pressuposto sobre um recurso r. Consideram-se modelos que definam: (i) para qualquer sistema S, o conjunto das suas execuções $[\![S]\!] = \{\mathcal{E} : \mathcal{E}$ é uma execução de $S\}$, sendo $[\![S]\!]$ um subconjunto do conjunto $EXEC$ que contém todas as execuções possíveis de qualquer sistema (isto é, $[\![S]\!] \subseteq EXEC$); e (ii) um conjunto $\models \varphi_r$, tal que $\models \varphi_r \subseteq EXEC$ é o subconjunto de todas as execuções possíveis que satisfazem o pressuposto φ_r. Usaremos $\mathcal{E} \models \varphi_r$ para representar a situação em que o pressuposto φ_r não é violado durante a execução $\mathcal{E}$.

No contexto destes modelos, a segurança-contra-exaustão é definida da seguinte forma.

Definição 2.1. *Um sistema S é seguro contra a exaustão de r em relação a um dado φ_r se e só se $\forall \mathcal{E} \in [\![S]\!] : \mathcal{E} \models \varphi_r$.*

Observe-se que esta formulação permite o estudo da segurança-contra-exaustão de um sistema em relação a diferentes tipos de pressupostos φ_r sobre um determinado recurso r.

2.1 Modelo de Exaustão de Recursos

O objectivo principal do Modelo de Exaustão de Recursos (REX) é permitir analisar de que forma é que a segurança-contra-exaustão pode ser afectada por diferentes combinações de pressupostos temporais e de pressupostos de faltas.

O modelo define dois intervalos temporais relacionados com a execução de um sistema e com o tempo necessário para exaurir um recurso, representados respectivamente por: *tempo de execução* e *tempo de exaustão*. O tempo de exaustão diz respeito a um pressuposto específico φ_r sobre um recurso específico r. Logo, daqui por diante, $[\![S]\!]$ representa o conjunto de execuções de um sistema S para um pressuposto fixo φ_r sobre um recurso específico r.

Definição 2.2. *Uma execução (de um sistema) $\mathcal{E}$ é um par $\langle T^{\mathcal{E}}_{exec}, T^{\mathcal{E}}_{exh} \rangle$, onde*

- $T^{\mathcal{E}}_{exec} \in \Re^+_0$ *e representa o tempo total de execução;*
- $T^{\mathcal{E}}_{exh} \in \Re^+_0$ *e representa o tempo necessário, desde o início da execução, para o pressuposto φ_r ser violado.*

Definição 2.3. *O pressuposto φ_r não é violado durante a execução $\mathcal{E}$, representado por $\mathcal{E} \models \varphi_r$, se e só se $T^{\mathcal{E}}_{exec} < T^{\mathcal{E}}_{exh}$.*

Combinando as Definições 2.1 e 2.3, é possível derivar a definição de um sistema seguro contra a exaustão de r.

Proposição 2.4. *Um sistema S é seguro contra a exaustão de r em relação a um determinado pressuposto φ_r se e só se $\forall \mathcal{E} \in [\![S]\!] : T^{\mathcal{E}}_{exec} < T^{\mathcal{E}}_{exh}$.*

A proposição estabelece que um sistema é seguro contra a exaustão de r se e só se a exaustão de recursos (isto é, a violação do pressuposto φ_r) não ocorrer durante nenhuma execução. Observe-se que mesmo que o sistema não seja seguro contra exaustão, isso não significa que o sistema falhe imediatamente a seguir à exaustão de recursos. Na realidade, um sistema pode apresentar um comportamento correcto entre o momento de exaustão e o fim da execução. Logo, um sistema não seguro contra exaustão pode executar correctamente durante todo o seu tempo de vida. No entanto, após a exaustão de recursos não há qualquer *garantia* que não ocorra uma falha por

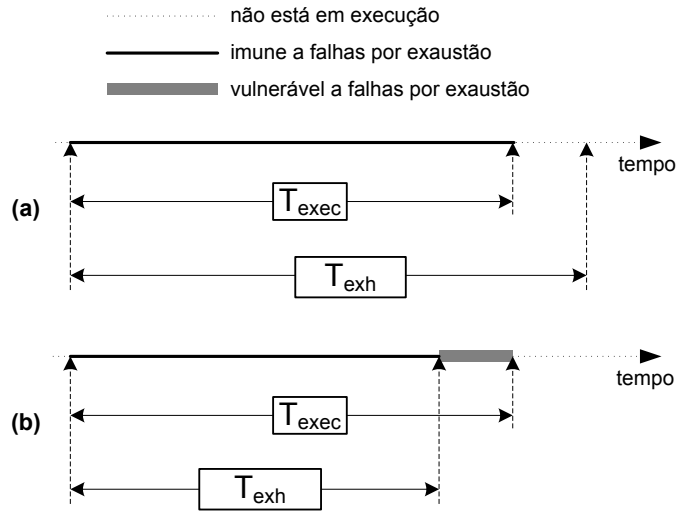

Figura 1: (a) Uma execução que não viola φ_r; (b) Uma execução que viola φ_r.

exaustão (isto é, uma falha provocada pela exaustão de recursos). A Figura 1 ilustra as diferenças entre uma execução de um sistema (potencialmente) seguro contra exaustão e uma execução "má" de um sistema não seguro contra exaustão. Um sistema seguro contra exaustão é imune a falhas por exaustão. Um sistema não seguro contra exaustão tem pelo menos uma execução (tal como a que é apresentada na Figura 1b) com um período de vulnerabilidade a falhas por exaustão (a parte sombreada da linha de tempo) onde o recurso está exaurido e portanto a correcção pode ser comprometida.

Num sistema distribuído tolerante a faltas, os nós são recursos importantes, de tal forma que tipicamente se faz o pressuposto sobre o número máximo de nós f que podem falhar durante a execução do sistema, e o sistema é projectado de forma a resistir um máximo de f falhas de nós. Este tipo de sistemas pode ser analisado segundo o modelo REX, colocando os nós como recursos e o pressuposto φ_{node} sendo igual a $n_{fail} \leq f$, onde n_{fail} representa o número de nós que, durante uma execução, estão falhados ao mesmo tempo. Por outras palavras, este pressuposto significa que um máximo de f nós podem estar falhados ao mesmo tempo.

Observe-se que num sistema onde os nós falhados não recuperam, este pressuposto é equivalente a assumir que durante uma execução não podem ocorrer mais do que f falhas. De acordo com a Proposição 2.4, um sistema distribuído tolerante a faltas cujos nós falhados não recuperem é seguro contra a exaustão de nós se e só se todas as execuções terminam antes do tempo necessário à produção de $f + 1$ falhas. De forma a construir-se um sistema tolerante a faltas seguro contra exaustão, seria necessário prever o número máximo N_{fail} de falhas possíveis de acontecer durante uma execução de forma a que o sistema fosse projectado para tolerar pelo menos $f = N_{fail}$ falhas.

Tal como a Secção 3 vai explicar em mais detalhe, o aspecto chave do estudo deste modelo é que a condição $T^{\mathcal{E}}_{exec} < T^{\mathcal{E}}_{exh}$ pode ser avaliada, isto é, existe a possibilidade de se determinar se a condição é mantida ou não, dependendo do tipo de pressupostos. Observe-se que a ideia não é saber os valores exactos de $T^{\mathcal{E}}_{exec}$ e $T^{\mathcal{E}}_{exh}$, mas sim raciocinar sobre os limites que lhes podem ser impostos, derivados do ambiente de execução e/ou dos pressupostos algorítmicos.

Desta forma, é possível prever em tempo de projecto do sistema ou mesmo em tempo de projecto dos algoritmos (usados pelo sistema), se o sistema pode ser seguro contra exaustão de acordo com os pressupostos em que se baseia. Mais genericamente, e como veremos mais à frente, é possível também fazer proposições sobre a segurança-contra-exaustão de modelos de sincronia

e de faltas, isto é, proposições sobre a resiliência potencial de algoritmos muito antes dos sistemas serem construídos. Como este objectivo em mente, começamos por definir duas propriedades cruciais do modelo, que derivam das definições anteriores.

Propriedade 2.5. *Uma* condição suficiente *para S ser seguro contra a exaustão de r em relação a um pressuposto φ_r é*
$\exists T_{exec_{max}} \in \Re_0^+ (\forall \mathcal{E} \in [\![S]\!] : T^{\mathcal{E}}_{exec} \leq T_{exec_{max}}) \wedge (\forall \mathcal{E} \in [\![S]\!] : T^{\mathcal{E}}_{exh} > T_{exec_{max}})$

Propriedade 2.6. *Uma* condição necessária *para S ser seguro contra a exaustão de r em relação a um pressuposto φ_r é*
$\exists T_{exh_{max}} \in \Re_0^+ (\forall \mathcal{E} \in [\![S]\!] : T^{\mathcal{E}}_{exh} \leq T_{exh_{max}}) \Rightarrow (\forall \mathcal{E} \in [\![S]\!] : T^{\mathcal{E}}_{exec} < T_{exh_{max}})$

A Propriedade 2.5 estabelece que um sistema S é seguro contra a exaustão de r em relação a um pressuposto φ_r se existe um limite superior $T_{exec_{max}}$ ao tempo de execução do sistema, e se o tempo de exaustão de qualquer execução é superior a $T_{exec_{max}}$.

A Propriedade 2.6 estabelece que um sistema S só pode ser seguro contra a exaustão de r em relação a um pressuposto φ_r se, dado um limite superior $T_{exh_{max}}$ ao tempo de exaustão do sistema, o tempo de execução de qualquer execução é inferior a $T_{exh_{max}}$.

3 Segurança-Contra-Exaustão vs Pressupostos de Sincronia

Esta secção analisa o impacto dos pressupostos de sincronia no projecto de sistemas seguros contra exaustão.

3.1 Sistemas Síncronos

Os sistemas desenvolvidos sob o modelo síncrono são relativamente fáceis de descrever. Este modelo tem três propriedades principais: o tempo de processamento local de qualquer operação tem um limite de tempo conhecido, as mensagens são entregues dentro de um intervalo de tempo conhecido e os relógios locais têm uma taxa de desvio limitada e conhecida em relação ao tempo real [HT94].

Se considerarmos um sistema síncrono S com um tempo de vida limitado sob o modelo REX, então é possível usar os limites de tempo inerentes ao modelo para estabelecer as condições de segurança-contra-exaustão em relação a um dado recurso r e pressuposto φ_r.

Corolário 3.1. *Se S é um sistema síncrono com um tempo de vida limitado $T_{exec_{max}}$ (isto é, $\forall \mathcal{E} \in [\![S]\!] : T^{\mathcal{E}}_{exec} \leq T_{exec_{max}}$) e $\forall \mathcal{E} \in [\![S]\!] : T^{\mathcal{E}}_{exh} > T_{exec_{max}}$, então S é seguro contra a exaustão de r em relação a φ_r.*

Proof: Deriva trivialmente da Propriedade 2.5. ■

Portanto, se o objectivo é conceber um sistema síncrono seguro contra exaustão e o sistema tem um tempo de vida limitado por $T_{exec_{max}}$, é preciso garantir que não é possível ocorrer qualquer exaustão durante $T_{exec_{max}}$. Por exemplo, num sistema distribuído tolerante a f faltas, isto significaria que não poderiam ocorrer mais do que f falhas de nós durante o intervalo de tempo definido por $T_{exec_{max}}$.

Observe-se que o Corolário 3.1 só se aplica a sistemas síncronos com um tempo de vida limitado. Um sistema síncrono pode contudo ter um tempo de vida ilimitado. Isto parece ser contraditório à primeira vista e por isso merece uma explicação mais detalhada. Um sistema síncrono é tipicamente composto por um conjunto de rondas (síncronas) com um tempo de execução limitado (por exemplo, num servidor síncrono que responde a pedidos de clientes, cada par pedido-resposta é uma ronda). Contudo, o número de rondas não é necessariamente limitado. Considera-se que um

sistema síncrono tem um tempo de vida limitado se o número de rondas é limitado. Caso contrário, o sistema tem um tempo de vida ilimitado. Se o sistema tiver um tempo de vida ilimitado e T_{exh} for limitado, então é possível provar o seguinte.

Corolário 3.2. *Se S é um sistema síncrono com um tempo de vida ilimitado (isto é, $\nexists T_{exec_{max}} \in \Re_0^+, \forall \mathcal{E} \in [\![S]\!] : T_{exec}^{\mathcal{E}} \leq T_{exec_{max}}$) e $\exists T_{exh_{max}} \in \Re_0^+, \forall \mathcal{E} \in [\![S]\!] : T_{exh}^{\mathcal{E}} \leq T_{exh_{max}}$, então S não é seguro contra a exaustão de r em relação a φ_r.*

Proof: Se o conjunto $\{T_{exec}^{\mathcal{E}} : \mathcal{E} \in [\![S]\!]\}$ não tem um limite superior, é impossível garantir que $T_{exec}^{\mathcal{E}} < T_{exh_{max}}$, para todas as $\mathcal{E} \in [\![S]\!]$ e, portanto, pela Propriedade 2.6, S não é seguro contra a exaustão de r. ■

De facto, os sistemas síncronos podem sofrer faltas acidentais ou maliciosas. Estas faltas podem originar dois tipos de efeitos: provocar falhas temporais que aumentam o tempo de execução esperado; provocar degradação dos recursos, por exemplo, falhas de nós e assim diminuir o valor de T_{exh}. Observe-se que ambos os efeitos forçam as condições do Corolário 3.2. Logo, num sistema síncrono, um adversário pode não só desencadear ataques que exaurem os recursos, como pode também violar os pressupostos temporais, mesmo que durante um intervalo limitado, ganhando tempo até os recursos serem exauridos. Consequentemente, o Corolário 3.2 formaliza e explica a crença actual da comunidade de investigação da área de segurança: os sistemas síncronos são frágeis e portanto os sistemas seguros devem ser construídos sob o modelo assíncrono.

3.2 Sistemas Assíncronos

A característica marcante de um sistema assíncrono é a inexistência de pressupostos temporais, logo os tempos de processamento local e de entrega de mensagens podem ser arbitrários e os relógios locais podem ter uma taxa de desvio ilimitada [FLP85, Lyn96]. Este modelo é bastante atractivo porque permite a concepção de programas e componentes que são mais fáceis de colocar a funcionar em diferentes ambientes.

Se considerarmos um sistema distribuído assíncrono S sob o modelo REX, então é impossível determinar quando é que S termina as suas execuções. Por outras palavras, o tempo de execução é ilimitado. Portanto, para percebermos se S é seguro contra a exaustão de r, para um determinado r e φ_r, é preciso analisar a relação entre T_{exec} e T_{exh}.

Será que um sistema distribuído assíncrono S pode ser seguro contra a exaustão de um recurso r? Apesar da arbitrariedade de T_{exec}, a condição $T_{exec}^{\mathcal{E}} < T_{exh}^{\mathcal{E}}$ tem de ser sempre mantida. Dado que $T_{exec}^{\mathcal{E}}$ pode ter um valor arbitrário, impossivel de saber através de cálculos apriorísticos, o sistema deveria ser construído de forma a assegurar que, em todas as execuções, $T_{exh}^{\mathcal{E}}$ é maior do que $T_{exec}^{\mathcal{E}}$. Isto é muito difícil de se conseguir para alguns tipos de recursos r e pressupostos φ_r. Um exemplo é assegurar que nunca falham mais do que f nós. Este trabalho apresenta uma solução para este caso particular na Secção 4. A solução é baseada numa arquitectura de sistema híbrida que garante segurança-contra-exaustão através de um subsistema parcialmente síncrono que executa rejuvenescimentos periódicos.

Se assumirmos que o sistema é homogeneamente assíncrono, e que o conjunto $\{T_{exh}^{\mathcal{E}} : \mathcal{E} \in [\![S]\!]\}$ tem um limite superior, é possível provar o seguinte corolário da Propriedade 2.6, semelhante ao Corolário 3.2:

Corolário 3.3. *Se S é um sistema assíncrono (e, portanto, $\nexists T_{exec_{max}} \in \Re_0^+, \forall \mathcal{E} \in [\![S]\!] : T_{exec}^{\mathcal{E}} \leq T_{exec_{max}}$) e $\exists T_{exh_{max}} \in \Re_0^+, \forall \mathcal{E} \in [\![S]\!] : T_{exh}^{\mathcal{E}} \leq T_{exh_{max}}$, então S não é seguro contra a exaustão de r em relação a φ_r.*

Proof: Deriva trivialmente do Corolário 3.2. ■

Este corolário é genérico, no sentido em que se aplica a qualquer tipo de sistema com um T_{exh} limitado para algum pressuposto φ_r. Contudo, as suas implicações em sistemas distribuídos tolerantes a faltas merecem um análise especial, dado que o presente trabalho foca a segurança-contra-exaustão deste tipo de sistemas.

Apesar dos sistemas distribuídos reais que trabalham sob o modelo assíncrono terem um T_{exh} limitado em termos de falhas de nós, a verdade é que têm vindo a ser usados com sucesso há já bastantes anos. Tal acontece porque, até muito recentemente, apenas as faltas acidentais (por exemplo, paragem, omissão) eram uma ameaça para os sistemas. Este tipo de faltas, sendo de natureza acidental, ocorre de uma forma aleatória. Portanto, se o ambiente em que o sistema executa for estudado em detalhe e se o sistema for concebido de uma forma inteligente (por exemplo, estimando um limite conservador para o T_{exec} que se aplique a um grande número de execuções), é possível obter-se um sistema assíncrono que se comporta como se fosse seguro contra exaustão, com uma probabilidade tão alta quanto se queira. Isto é, apesar do sistema ter o síndroma de falha explicado acima, seria bastante difícil observá-lo na prática.

Contudo, quando começamos a considerar faltas maliciosas, é necessário raciocinar de forma diferente. Este tipo de faltas é intencional (não acidental) e portanto a sua distribuição não é aleatória: a distribuição real pode ser moldada de acordo com a vontade do adversário cujo objectivo principal é afectar o funcionamento do sistema (por exemplo, forçar o sistema a executar durante mais tempo do que qualquer limite estimado para T_{exec}). Nestas condições, ter um T_{exh} limitado (que se obtém por exemplo quando se usa um limite estacionário para o número de falhas de nós) implica muito provavelmente a falha real do sistema devido a uma falha de exaustão.

Em suma, T_{exh} não deve ter um limite superior num sistema distribuído assíncrono tolerante a faltas que opere num ambiente susceptível a faltas maliciosas (intencionais).

Os resultados apresentados nesta secção permitem tirar duas conclusões. Em primeiro lugar, o impacto teórico destes resultados é independente do tipo de faltas. Isto é, os síndromas de falha aqui descritos eram desconhecidos anteriormente e mesmo com faltas acidentais podem causar a falha inesperada de sistemas distribuídos síncronos e assíncronos. Consequentemente, estes resultados podem alertar outros investigadores e ajudar a conceber melhores sistemas distribuídos.

Em segundo lugar, o impacto prático dos mesmos resultados pode ser cada vez maior, na medida em que os sistemas, críticos ou genéricos, estão a tornar-se presas de ataques de piratas informáticos (faltas maliciosas). Isto significa que, como probabilidade crescente, os sistemas com o síndroma de falha aqui descrito (modelo assíncrono + tempo de exaustão limitado) não apenas podem, como serão mesmo atacados e feitos falhar.

4 Resiliência Proactiva

Uma das abordagens mais interessantes para se evitar a exaustão de recursos provocada pela corrupção acidental ou maliciosa de componentes é a recuperação proactiva [OY91], que pode ser vista como uma forma de redundância dinâmica [SS92]. O objectivo deste mecanismo é conceptualmente simples — os componentes são rejuvenescidos periodicamente para se remover os efeitos de faltas/ataques maliciosos. Se o rejuvenescimento for feito de forma frequente, um adversário é incapaz de corromper recursos suficientes para afectar o funcionamento do sistema. A recuperação proactiva tem vindo a ser proposta em bastantes contextos: refrescamento de chaves criptográficas [HJKY95, HJJ+97, GGJR00, ZSR05, CKLS02, ZSvR02, MS04], restauro do código de sistema de uma fonte segura para se eliminar os efeitos de ataques [OY91, CL02], substituição de componentes para remoção de vulnerabilidades existentes em versões anteriores (por exemplo, erros de software que poderiam parar o sistema ou que poderiam ser explorados por um atacante).

Intuitivamente, se usarmos uma estratégia bem planeada para a recuperação proactiva, T_{exh} pode ser aumentado recorrentemente de forma a que seja sempre superior a T_{exec} em todas as execuções. Contudo, esta intuição é difícil de substanciar se o sistema for assíncrono. A simples tarefa de lançar atempadamente um procedimento periódico de recuperação é impossível de se conseguir sob o modelo assíncrono, nomeadamente se estiver sujeito a faltas maliciosas que podem atrasar deliberadamente a execução do procedimento de recuperação. Este raciocínio e o Corolário 3.3 permitem-nos concluir que não é possível garantir a segurança-contra-exaustão de um sistema assíncrono com tempo de exaustão limitado através de recuperação proactiva assíncrona.

A impossibilidade de se construir um sistema distribuído assíncrono tolerante a f faltas arbitrárias seguro contra exaustão, nomeadamente na presença de faltas maliciosas, e mesmo fazendo uso de recuperação proactiva assíncrona, conduziu-nos à investigação de modelos híbridos para a recuperação proactiva.

4.1 Modelo de Resiliência Proactiva

A recuperação proactiva é útil para rejuvenescer periodicamente componentes e remover os efeitos de faltas/ataques, desde que ofereça garantias temporais. De facto, o resto do sistema pode ser completamente assíncrono, apenas o mecanismo de recuperação proactiva necessita de execução síncrona. Este tipo de requisito indica que uma possível abordagem para se usar a recuperação proactiva de forma efectiva é modelando-a e arquitectando-a sob um modelo de sistema híbrido.

Neste contexto, propõe-se o Modelo de Resiliência Proactiva (PRM), uma abordagem mais resiliente à recuperação proactiva. O PRM define um sistema aumentado com recuperação proactiva através de um modelo composto por duas partes: o subsistema de recuperação proactiva e o sistema principal, o último sendo recuperado proactivamente pelo primeiro. Cada uma destas duas partes faz diferentes pressupostos temporais e diferentes pressupostos de faltas, e deve ser concebido de forma a assegurar que os pressupostos se verificam na prática.

O sistema principal executa as aplicações e os protocolos "normais". Desta forma, os modelos de sincronia e de faltas do sistema principal dependem das aplicações/protocolos que executem nesta parte do sistema. Por exemplo, o sistema principal pode operar num ambiente assíncrono com faltas maliciosas. O subsistema de recuperação proactiva executa os protocolos de recuperação proactiva que rejuvenescem as aplicações/protocolos do sistema principal. Este subsistema, apesar de ser simples em termos de funcionalidades. é mais exigente em termos de pressupostos temporais e de faltas, e é modelado por um componente abstracto distribuído designado por *Monitor de Recuperação Periódica* (PRW). O componente é abstracto porque admite diferentes instanciações. Tipicamente, uma instanciação específica é escolhida de acordo com a aplicação/protocolo concretos que precisam de ser recuperados proactivamente.

A arquitectura de um sistema com um PRW é ilustrada na Figura 2. Cada nó tem um módulo local, designado por *PRW local*. Estes módulos estão organizados em agrupamentos, designados por *agrupamentos PRW*, e os PRWs locais em cada agrupamento estão interligados por uma *rede de controlo* síncrona e segura.

Conceptualmente, um PRW local é um módulo separado do sistema operativo. Na prática, esta separação entre o PRW local e o sistema operativo pode ser conseguida de duas formas distintas: (1) o PRW local é concretizado num módulo de hardware separado *tamper-proof* (por exemplo, um smartcard, ou uma placa PCI [Ken80, Tru04]) e desta forma a separação é física; (2) o PRW local é concretizado no hardware nativo, com uma separação virtual (por exemplo, usando virtualização de software [BDF+03]) entre o PRW local e os processos do sistema operativo.

A forma como os agrupamentos são organizados depende dos requisitos de rejuvenescimento. Tipicamente, um agrupamento é composto por nós que são de alguma forma interdependentes em

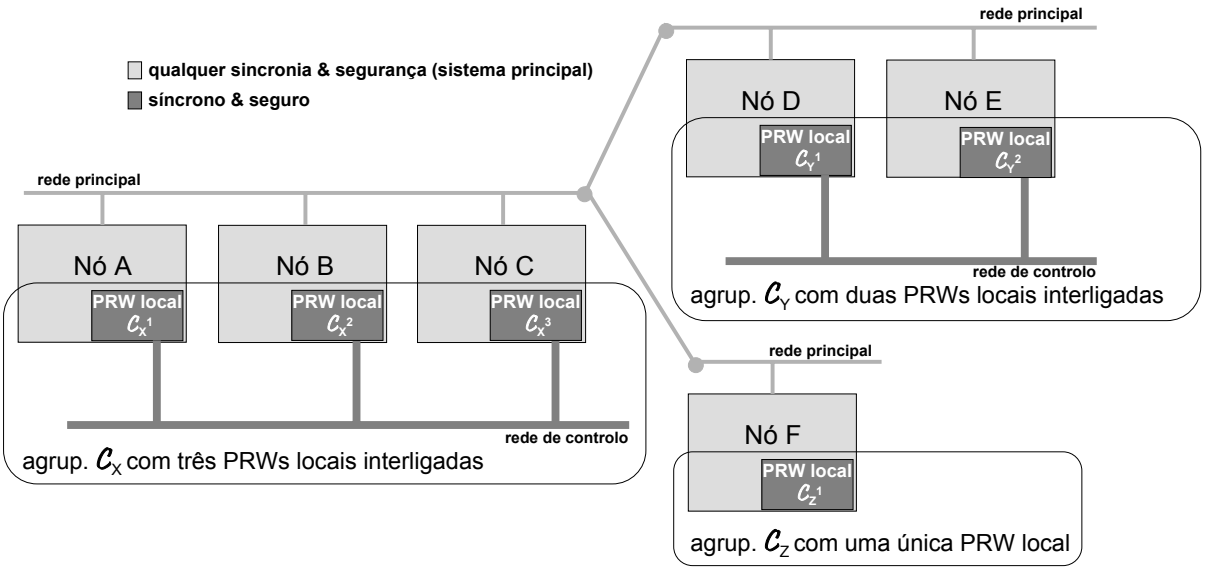

Figura 2: A arquitectura de um sistema com um PRW.

termos de rejuvenescimento (por exemplo, precisam de trocar informação durante a recuperação). Neste trabalho focamos duas configurações específicas:

PRW^l é composta por n agrupamentos, cada um composto por um único PRW local. Portanto, cada agrupamento de uma configuração PRW^l é exactamente como o agrupamento $\mathcal{C}_Z$ apresentado na Figura 2, e, consequentemente, não existe qualquer rede de controlo em todos os agrupamentos;

PRW^d é composta por um agrupamento único, por sua vez composto por todos os PRWs locais. Por exemplo, considere-se um sistema composto por 3 nós. Se este sistema estivesse organizado de acordo com a configuração PRW^d, então o (único) agrupamento seria como o agrupamento $\mathcal{C}_X$ apresentado na Figura 2. Neste caso, cada PRW local estaria interligado através da mesma rede de controlo.

A configuração PRW^l deve ser usada em cenários onde o procedimento de recuperação requeira apenas informação local e, portanto, não exista a necessidade de execução distribuída. A configuração PRW^d deve ser usada quando a recuperação é feita através de um procedimento de recuperação completamente distribuído em que cada PRW local precisa de participar. Existem muitas mais configurações possíveis, nomeadamente configurações compostas por agrupamentos heterogéneos, mas não são abordadas neste trabalho.

4.2 Rejuvenescimento Periódico Atempado

O PRW executa rejuvenescimentos periódicos através de um serviço de execução periódica atempada. Esta secção define o serviço de execução periódica atempada, propõe um algoritmo para o concretizar, e especifica as garantias de tempo-real requeridas pelo PRW. Seguidamente, a Secção 4.3 prova que os sistemas aumentados com um PRW são seguros contra a exaustão de nós desde que sejam cumpridas um conjunto de condições.

Cada agrupamento PRW executa a sua própria instância do serviço de execução periódica atempada, e não há qualquer restrição em termos da coordenação das diferentes instâncias. Apesar de serem independentes, cada agrupamento oferece o mesmo conjunto de propriedades ditadas

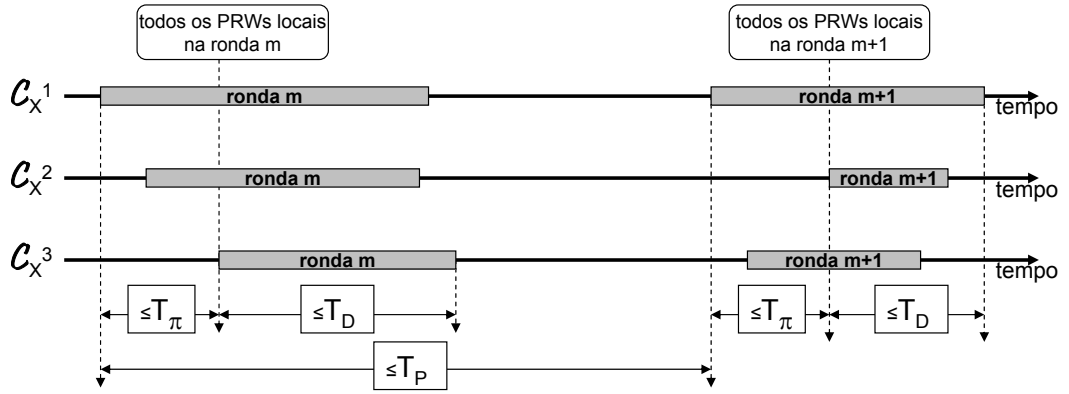

Figura 3: Relação entre T_P,T_D e T_π num agrupamento $\mathcal{C}_X$ com três PRWs locais.

por quatro parâmetros globais: F, T_P, T_D e T_π. Nomeadamente, cada agrupamento executa um procedimento de rejuvenescimento F em rondas, e cada ronda inicia a sua execução a menos de T_P do início de execução da ronda anterior. Este início de execução é feito por pelo menos um PRW local (em cada agrupamento) e todos os PRWs locais (do mesmo agrupamento) iniciam a execução da mesma ronda com uma diferença máxima de T_π. Desta forma, o T_π corresponde ao intervalo de tempo máximo entre o instante em que um PRW local começa a executar uma certa ronda e o instante em que todos os outros PRWs locais começam a executar a mesma ronda. Para além disso, cada agrupamento garante que, logo que todos os PRWs locais estejam na mesma ronda, o tempo de execução de F é limitado por T_D. Portanto, o pior tempo de execução possível para cada ronda de F é dado por $T_\pi + T_D$. A Figura 3 ilustra a relação entre T_P, T_D, e T_π, num agrupamento com três PRWs locais. De seguida é apresentada uma definição formal do serviço de execução periódica atempada.

Definição 4.1. *Seja F um procedimento e $T_D, T_P, T_\pi \in \Re_0^+$, tal que $T_D + T_\pi < T_P$. Um conjunto de componentes $\mathcal{C}$, organizados em s agrupamentos disjuntos e não vazios $\mathcal{C}_1, ..., \mathcal{C}_s$, oferece um* serviço de execução periódica atempada $\langle F, T_D, T_P, T_\pi \rangle$*, se e só se:*

1. *os componentes de cada agrupamento $\mathcal{C}_i$ executam F em rondas, e portanto F é um procedimento distribuído em cada agrupamento;*

2. *para cada instante de tempo real t do tempo de execução de $\mathcal{C}$, existe uma ronda de F que inicia a sua execução em cada agrupamento $\mathcal{C}_i$ a menos de T_P de t, isto é, pelo menos um componente C em cada agrupamento $\mathcal{C}_i$ inicia a execução de uma ronda de F a menos de T_P de t;*

3. *todos os componentes de um agrupamento $\mathcal{C}_i$ iniciam a execução da mesma ronda de F a menos de T_π de qualquer outro componente do mesmo agrupamento;*

4. *cada agrupamento $\mathcal{C}_i$ garante que, logo que todos os componentes estejam na mesma ronda de F, o tempo de execução de F é limitado superiormente por T_D.*

Corolário 4.2. *Se $\mathcal{C}$ é um conjunto de componentes, organizados em s agrupamentos $\mathcal{C}_1, ..., \mathcal{C}_s$, que oferece um serviço de execução periódica atempada $\langle F, T_D, T_P, T_\pi \rangle$ então, para cada instante de tempo real t do tempo de execução de $\mathcal{C}$, existe uma ronda de F, por cada agrupamento, com início de execução a menos de T_P de t e que termina a execução a menos de $T_P + T_D + T_\pi$ de t.*

Definição 4.3. *Um sistema aumentado com um $PRW(\langle F, T_D, T_P, T_\pi\rangle)$ tem um PRW local em cada nó. Os PRWs locais estão organizados em agrupamentos e em conjunto oferecem o serviço de execução periódica atempada $\langle F, T_D, T_P, T_\pi\rangle$.*

Tal como referido anteriormente, o PRW admite duas configurações de agrupamentos particulares — PRW^l and PRW^d. Estas configurações são definidas da seguinte forma.

Definição 4.4. *Um sistema aumentado com um $PRW^l(\langle F, T_D, T_P, T_\pi\rangle)$ é um sistema aumentado com um $PRW(\langle F, T_D, T_P, T_\pi\rangle)$ tal que existem n agrupamentos $\mathcal{C}_1, ..., \mathcal{C}_n$, e cada agrupamento $\mathcal{C}_i$ é composto por um único PRW local.*

Definição 4.5. *Um sistema aumentado com um $PRW^d(\langle F, T_D, T_P, T_\pi\rangle)$ é um sistema aumentado com um $PRW(\langle F, T_D, T_P, T_\pi\rangle)$ tal que existe um único agrupamento $\mathcal{C}_1$ composto por todos os PRWs locais.*

Um serviço de execução periódica atempada pode ser construído usando, por exemplo, o Algoritmo 1, num ambiente que garanta as seguintes propriedades:

P1 Existe um limite superior conhecido ao tempo de processamento de todos os PRWs locais.

P2 Existe um limite superior conhecido à taxa de desvio do relógio de cada PRW local.

P3 Existe um limite superior conhecido ao tempo de entrega das mensagens enviadas pela rede de controlo que interliga os PRWs locais de um mesmo agrupamento.

Considere-se então que cada PRW local executa o Algoritmo 1, onde a função *relogio* retorna o valor actual do relógio do PRW local, F é o procedimento de recuperação que deve ser executado periodicamente de forma atempada e T_P é a periodicidade de recuperação desejada. O valor δ define um intervalo temporal de segurança de forma a garantir que qualquer recuperação é iniciada a menos de T_P da recuperação imediatamente anterior, na presença dos limites superiores assumidos para o tempo de processamento local (P1) e para a taxa de desvio dos relógios (P2). Observe-se que entre a instrução $wait$ na linha 2 e o início de execução de F na linha 7, existe um conjunto de instruções que levam tempo (limitado) a executar. O valor de δ deve garantir que as recuperações consecutivas nunca distam mais do que T_P independentemente do tempo de execução real destas instruções, e tendo em conta a taxa de desvio máximo do relógio. No entanto, o valor de δ deve também garantir que todas as PRWs locais iniciam a execução de F a menos de T_π de qualquer outra. Logo, o valor de δ não deve ser superior a $T_P - (T_D + T_\pi)$ de forma a garantir-se que o PRW local ${\mathcal{C}_i}^1$ em cada agrupamento $\mathcal{C}_i$ não inicia a execução de F demasiado cedo (isto é, quando outros PRWs locais podem ainda estar a executar a ronda anterior). Nestas condições, o algoritmo garante que F é sempre iniciado, em cada agrupamento $\mathcal{C}_i$, pelos PRWs locais ${\mathcal{C}_i}^1$ a menos de T_P do último início de execução. Para além disso, uma vez que se garante que as diferentes rondas não se intersectam, os instantes de início de execução de F nos PRWs locais de um mesmo agrupamento, diferem no máximo na soma do tempo máximo de entrega de mensagens (P3) e do tempo máximo de processamento, isto é, o tempo necessário para a mensagem *inicia* ser entregue e processada por todos os PRWs locais. Por conseguinte, o valor de T_π é definido por esta soma. Nesta situação, cada PRW local oferece um serviço de execução periódica atempada $PRW(\langle F, T_D, T_P, T_\pi\rangle)$ desde que garanta que, a partir do momento que todos os PRWs locais estão na mesma ronda de F, o seu tempo de execução é limitado por T_D.

Algoritmo 1: Serviço de execução periódica atempada executado por cada PRW local $\mathcal{C}_i{}^j$ no agrupamento $\mathcal{C}_i$

initialization: $t_{ultimo} \leftarrow relogio$

begin

  while *true* **do**

    /* PRWs locais $\mathcal{C}_i{}^j$ com $j = 1$ em cada agrupamento $\mathcal{C}_i$ coordenam o processo de recuperação */

1 **if** $j = 1$ **then**

2 wait until $relogio = t_{ultimo} + T_P - \delta$

3 $t_{ultimo} \leftarrow relogio$

4 multicast(envia, $\mathcal{C}_i$)

5 **else**

6 receive(envia)

7 execute F

end

4.3 Construção de Sistemas Seguros Contra a Exaustão de Nós

Um sistema distribuído aumentado com um $PRW(\langle F, T_D, T_P, T_\pi \rangle)$ pode ser tornado seguro contra a exaustão de nós em certas condições, tal como vai ser provado no Teorema 4.6. Este teorema estabelece que, se for possível definir um limite inferior para o tempo de exaustão (isto é, o tempo necessário para se produzir $f + 1$ falhas de nós) de qualquer execução através de uma constante conhecida $T_{exh_{min}}$, então a segurança contra a exaustão de nós é conseguida desde que $T_P + T_D + T_\pi < T_{exh_{min}}$.

No que se segue, seja $[\![S]\!]$ o conjunto de execuções de um sistema distribuído S tolerante a f faltas sob o modelo REX e considere o pressuposto $\varphi_{node} = n_{fail} \leq f$, onde n_{fail} representa o número de nós que, durante uma execução, estão falhados simultaneamente. Observe-se que o tipo de falha não é especificado, mas apenas que os nós podem falhar de alguma forma e que esta falha pode ser recuperada através da execução de um procedimento de rejuvenescimento. Uma falha de um nó pode ser por exemplo a divulgação de informação confidencial, ou uma intrusão de um pirata informático que compromete o comportamento de algumas partes do sistema. Observe-se também que o procedimento de rejuvenescimento depende do tipo específico de falha de que é necessário recuperar. Por exemplo, enquanto que uma intrusão pode implicar um reinício do sistema e o recarregamento do código e do estado do sistema operativo e das aplicações a partir de uma origem confiável, a divulgação de informação confidencial pode ser resolvida tornando essa informação obsoleta.

Teorema 4.6. *Suponha que:*

1. *S é um sistema composto por um total de n nós que, uma vez falhados, não recuperam e, seja $T_{exh_{min}} = \inf(\{T^{\mathcal{E}}_{exh} : \mathcal{E} \in [\![S]\!]\})$[1];*

2. *O tempo necessário à produção de $f + 1$ ($\leq n$) falhas de nós em qualquer instante é independente do número de nós que estão falhados em cada momento;*

3. *F é um procedimento distribuído que, após estar terminado, garante que todos os nós envolvidos na sua execução não estão falhados.*

[1] $\inf()$ denota o ínfimo de um conjunto de números reais, isto é, o maior limite inferior do conjunto.

Logo, o sistema S aumentado com o $PRW(\langle F, T_D, T_P, T_\pi\rangle)$ tal que $T_P + T_D + T_\pi < T_{exh_{min}}$ é seguro contra exaustão em relação ao pressuposto φ_{node}.

Demonstração: A assumpção (1) implica que, em qualquer execução de S, de um estado com *0* nós falhados, demora-se pelo menos $T_{exh_{min}}$ até serem produzidas $f + 1$ falhas de nós. Seja m um número natural tal que $m + f + 1 \leq n$. Então, usando a assumpção (2), podemos concluir que, em qualquer execução de S, demora-se pelo menos $T_{exh_{min}}$ a chegar a um estado com $m + f + 1$ nós falhados a partir de um estado com m nós falhados[2]. Isto também implica o seguinte:

4. em qualquer execução de S, o número de nós falhados durante o intervalo de tempo $]t, t + T_{exh_{min}}[$ é no máximo f.

Com vista à prova por contradição, assuma que existe uma execução do sistema S, aumentado com um a $PRW(\langle F, T_D, T_P, T_\pi\rangle)$ tal que $T_P+T_D+T_\pi < T_{exh_{min}}$, que viola o pressuposto φ_{node}. Isto significa que há um instante temporal t_C onde estão falhados mais do que f nós. Observe-se que t_C não pode ocorrer a menos de $T_{exh_{min}}$ do instante inicial de execução do sistema, porque isto significaria que mais do que $f + 1$ falhas de nós tinham sido produzidas em menos de $T_{exh_{min}}$ a partir de um estado com *0* nós falhados, o que contradiz a assumpção (1). Logo, t_C ocorre num instante que dista pelo menos $T_{exh_{min}}$ do instante inicial de execução do sistema.

Logo, pela assumpção (4), no instante $t_I = t_C - T_{exh_{min}}$ há pelo menos um nó falhado, porque não é possível falharem mais do que f nós em menos de $T_{exh_{min}}$. Dado que a natureza do procedimento F é recuperar os nós do agrupamento onde F é executado (assumpção (3)), a execução de S aumentado com um $PRW(\langle F, T_D, T_P, T_\pi\rangle)$ onde $T_P+T_D+T_\pi < T_{exh_{min}}$ garante que qualquer nó que esteja falhado em t_I, é recuperado no pior caso no instante $t_I+T_P+T_D+T_\pi$ e, portanto, é recuperado mais cedo do que $t_C = t_I + T_{exh_{min}}$. Se um dos nós que está falhado em t_I recupera antes de t_C e há mais do que f nós falhados em $t_C = t_I + T_{exh_{min}}$, então mais do que f nós falharam no intervalo $]t_I, t_I + T_{exh_{min}}[$. Mas isto é contraditório com a assumpção (4). ■

A partir do Teorema 4.6 deduz-se que, de forma a construir-se um sistema tolerante a faltas seguro contra a exaustão de nós, o arquitecto de sistema deve escolher um grau apropriado f de tolerância a faltas, tal que $T_P + T_D + T_\pi < T^{\mathcal{E}}_{exh}$, para qualquer execução $\mathcal{E}$. Por outras palavras, um intervalo com tamanho $T_P + T_D + T_\pi$ não deve ser suficiente para que sejam produzidas $f + 1$ falhas de nós, durante todo o tempo de vida do sistema.

5 Replicação de Máquina de Estados Resiliente e Disponível

5.1 Motivação

Hoje em dia, uma das maiores preocupações acerca dos serviços providenciados por sistemas computacionais está relacionado com a sua disponibilidade. Isto aplica-se especialmente aos serviços oferecidos através da Internet. A construção de serviços altamente disponíveis envolve, por um lado, a concepção e a concretização de serviços correctos que sejam tolerantes a um largo conjunto de faltas, e por outro lado, a certificação de que o acesso aos serviços é sempre garantido com elevada probabilidade. Ambas as tarefas podem ser cumpridas através da aplicação de técnicas de replicação.

A replicação é uma forma conhecida de se melhorar a disponibilidade de um serviço: se um serviço pode ser acedido através de vários caminhos independentes, então a probabilidade de um cliente ser capaz de o usar é também maior. Mas a replicação tem custos, nomeadamente

[2]Observe-se que um nó pode falhar, ser recuperado, falhar outra vez, e assim sucessivamente. Portanto, o número total de falhas de nós não corresponde necessariamente ao número de nós que estão falhados em cada momento.

é necessário garantir uma coordenação correcta entre as várias réplicas. Para além disso, uma vez que a Internet é um ambiente imprevisível e inseguro, a coordenação entre as réplicas deve ser assegurada nas piores condições de funcionamento, isto é, a ausência de garantias sobre o tempo de entrega das mensagens trocadas entre as réplicas e a possibilidade de ocorrerem faltas arbitrárias despoletadas por adversários maliciosos. Muitos trabalhos anteriores debruçaram-se sobre técnicas de replicação que toleram faltas arbitrárias sob o modelo assíncrono. Todas estas técnicas fazem o pressuposto de que o número de réplicas faltosas é limitado por um valor conhecido [BT85, CR93, MR97b, MR97a, DGGS99, MR00, CKS00]. Contudo, sob o modelo assíncrono, este tipo de pressuposto pode ser problemático. Como foi demonstrado na Secção 3.2, não há forma de se assegurar que só ocorrerão no máximo f faltas durante a execução do sistema que oferece o serviço.

5.2 Replicação de Máquina de Estados

Uma máquina de estados é definida por um conjunto de variáveis e por um grupo de comandos. A colecção de variáveis define o estado do sistema. Os comandos são usados para efectuar modificações nas variáveis e/ou para produzir algum resultado (por exemplo, ler o valor de uma variável) [Lam78, Sch90]. Praticamente todos os programas de computador podem ser modelados como uma máquina de estados. Em particular, este trabalho foca as aplicações cliente-servidor, que também se encaixam neste modelo: o servidor é responsável por manter o estado e os clientes enviam comandos para modificar ou ler o estado. Esta forma de olhar para as aplicações cliente-servidor facilita o raciocínio sobre como tornar estas aplicações tolerantes a faltas. A forma mais simples de se construir uma aplicação cliente-servidor é concretizando um servidor único centralizado que processa todos os comandos enviados pelos clientes. Desde que o servidor não falhe, os comandos são processados pela ordem com que são recebidos dos clientes. Mas caso possam ocorrer faltas, esta aproximação centralizada não funciona. O servidor pode parar de funcionar e o sistema ficar indisponível, ou então, pior, o servidor pode ser comprometido por um adversário malicioso e ver o seu estado ser alterado de forma arbitrária. De forma a tolerar-se estes tipos de faltas, é necessário replicar o servidor. O grau de replicação depende do tipo (por exemplo, faltas por paragem, faltas arbitrárias) e da quantidade de faltas que é necessário tolerar. Existem alguns protocolos que permitem concretizar uma máquina de estados replicada tolerante a faltas de paragem [Ske82, OL88, Lam98] e tolerante a faltas arbitrárias [Rei95, CL02, ADD$^+$06]. Neste trabalho não é feita qualquer restrição ao tipo de faltas que pode acontecer — uma réplica pode falhar de forma arbitrária, quer por paragem, quer por corrupção do estado e/ou da lógica de execução.

O estado da arte actual permite construir aplicações cliente-servidor baseadas em máquina de estados replicada capazes de tolerar um número especificado f de faltas arbitrárias. No entanto, para que esta solução seja resiliente, é necessário garantir que nunca acontecem mais do que f faltas durante a execução do sistema. Neste contexto, propomos aplicar o Modelo de Resiliência Proactiva (PRM), apresentado na Secção 4.1, ao cenário da replicação de máquina de estados. O serviço de execução periódica atempada é usado para recuperar proactivamente as réplicas, garantindo-se que

- nunca são corrompidas mais do que f réplicas;
- a execução da máquina de estados replicada nunca é interoompida.

5.3 Monitor de Recuperação Proactiva para Máquina de Estados

Propomos o Monitor de Recuperação Proactiva para Máquina de Estados (SMW) como uma instanciação do PRW$^l\langle F, T_D, T_P, T_\pi\rangle$ apresentado na Secção 4.1. Isto significa que existe uma

réplica por agrupamento, e que não é usada qualquer rede de controlo. A Figura 4 ilustra a arquitectura de um sistema com um SMW. O objectivo do SMW é rejuvenescer periodicamente as réplicas de forma a que não mais do que f réplicas sejam comprometidas e a segurança contra a exaustão de nós seja assim garantida. Para além disso, as recuperações não devem afectar a disponibilidade global do sistema.

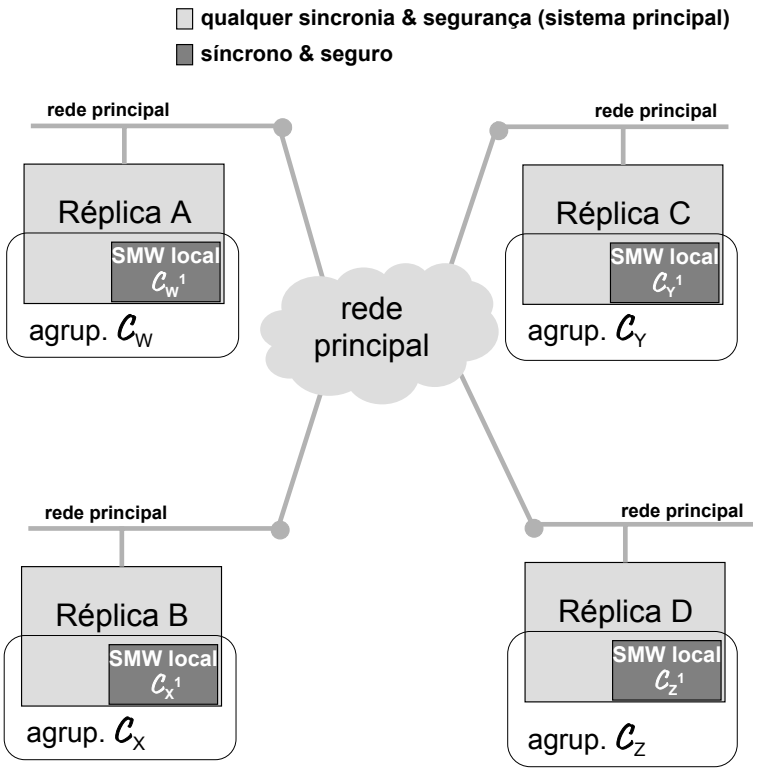

Figura 4: Arquitectura de uma máquina de estados replicada aumentada com um SMW.

O SMW executa o Algoritmo 1 (página 14) de forma a executar periodicamente e atempadamente o procedimento *refrescaCodigoEstado* apresentado no Algoritmo 2 (página 18). Observe-se que existe um único SMW local $\mathcal{C}_i{}^1$ em cada agrupamento $\mathcal{C}_i$. Desta forma, não é necessário enviar a mensagem na linha 4 do Algoritmo 1.

No Algoritmo 2, a linha 1 desliga o sistema operativo do sistema principal, e portanto pára a execução da máquina de estados local. A linha 2 verifica se o código do sistema operativo está corrompido. Para executar esta tarefa, pode guardar-se inicialmente uma assinatura do código do sistema operativo em memória que permita apenas leitura, e depois em cada recuperação compara-se a assinatura do código actual com a que está guardada. Na linha 3, o código do sistema operativo pode ser restaurado a partir de um dispositivo de armazenamento onde existe uma cópia limpa carregada previamente no início da execução do sistema. De forma a introduzir-se diversidade, podem usar-se diferentes versões do sistema operativo em cada recuperação. Nas linhas 4–5, o código da máquina de estados pode ser verificado e restaurado de forma similar ao código do sistema operativo. A linha 6 arranca o sistema operativo do sistema principal a partir de código limpo e a réplica é assim colocada num estado correcto. Assume-se que a máquina de estados local é iniciada automaticamente logo que o sistema operativo termina o processo de arranque.

Dado que o estado da máquina de estados local pode ter sido comprometido antes do rejuvenescimento, poderá ser necessário transferir um estado correcto das restantes réplicas. Assume-se que esta transferência/recuperação de estado é feito automaticamente pelo código da máquina de

Algoritmo 2: Procedimento *refrescaCodigoEstado* executado por cada SMW local.

```
   begin
   |  /* desliga o sistema operativo do sistema principal      */
1  |  desligaSO()
   |  /* restaura o código do sistema operativo se estiver
   |     corrompido                                            */
2  |  if código OS está corrompido then
3  |  |  restauraCodigoSO()
   |  /* restaura o código da máquina de estados se estiver
   |     corrompida                                            */
4  |  if código ME está corrompido then
5  |  |  restauraCodigoME()
   |  /* neste ponto, o sistema operativo e a máquina de estados
   |     podem ser iniciados porque o seu código está correcto
   |     */
6  |  iniciaSO()
   end
```

estados logo que (re)inicia. Em [CL02] é apresentado um mecanismo eficiente para se realizar salvaguardas e transferência de estado, concebido especificamente para máquinas de estado replicadas sujeitas a faltas arbitrárias. No entanto, dependendo das garantias temporais da rede principal, a recuperação de estado pode ter um tempo de execução ilimitado, uma vez que requer troca de informação através da rede principal. Dado que a rede principal é potencialmente assíncrona, as mensagens enviadas através dela podem ter um tempo de entrega ilimitado. Não obstante, é possível estimar um limite superior ao tempo de entrega que será satisfeito com elevada probabilidade em condições normais. No pior caso, se ocorrerem atrasos anormais, a disponibilidade da máquina de estados replicada pode ser afectada, mas a correcção é sempre preservada.

Depois do término do procedimento de rejuvenescimento *refrescaCodigoEstado* num SMW local, a réplica correspondente está correcta. O sistema é seguro contra exaustão de nós em relação a um determinado adversário se os rejuvenescimentos estiverem organizados de forma a que o adversário não tenha tempo suficiente para comprometer mais do que f réplicas entre rejuvenescimentos.

Por questões de espaço, não é aqui apresentada a demonstração formal de que é possível construir uma máquina de estados replicada segura contra a exaustão de nós fazendo uso do SMW. A ideia base é que o SMW oferece garantias temporais e por isso consegue garantir, por um lado, que as recuperações são executadas com uma determinada periodicidade e, por outro lado, que o procedimento de recuperação tem um tempo de execução limitado.

5.4 Segurança-Contra-Exaustão e Disponibilidade

Nesta secção é discutida a estratégia de recuperação que deve ser aplicada de forma a garantir-se que as recuperações necessárias à segurança-contra-exaustão não afectam a disponibilidade.

Uma solução trivial para se conseguir segurança contra a exaustão de nós seria rejuvenescer todas as réplicas ao mesmo tempo: as réplica seriam paradas simultaneamente, rejuvenescidas, e reiniciadas novamente. Dado que não existiria progresso do sistema durante o rejuvenescimento, apenas as réplicas previamente comprometidas teriam de restaurar o seu estado. O problema desta solução é que a máquina de estados replicada estaria indisponível durante o rejuvenescimento e

isto nem sempre é desejável. Contudo, em cenários onde a disponibilidade não é requisito, esta solução tem a vantagem de minimizar o número de transferências de estado, dado que apenas os estados corrompidos precisam de ser restaurados.

De forma a evitar-se a interrupção do serviço prestado pela máquina de estados replicada, é necessário fazer-se o seguinte:

1. Definir o número máximo de réplicas que podem recuperar simultaneamente (designemo-lo por k). Note que uma réplica em recuperação pode não processar pedidos de clientes até terminar a recuperação. Desta forma, uma réplica em recuperação pode ser considerada como se tivesse parada na perspectiva dos algoritmos de máquina de estados replicada (por exemplo, algoritmos de difusão atómica [DSU04]).

2. Implantar o sistema com um número de réplicas suficientes para tolerar f réplicas com faltas arbitrárias e k réplicas paradas. Neste contexto, este trabalho estabelece um novo limite inferior ao número n de réplicas necessárias para se construir uma máquina de estados replicada resiliente e disponível: $n \geq 3f + 2k + 1$.

5.4.1 Justificação para o Novo Limite $n \geq 3f + 2k + 1$

Classicamente, uma máquina de estados replicada com $3f + 1$ réplicas é capaz de tolerar até f faltas arbitrárias [PSL80]. Este limite está associado ao protocolo de difusão atómica usado para garantir a coerência da máquina de estados replicada. Se ocorrerem mais do que f faltas, tanto a correcção como o progresso do protocolo de difusão atómica podem ser afectados. Em particular, se ocorrerem $f+1$ faltas por paragem, o protocolo (tipicamente) bloqueia. Portanto, uma máquina de estados replicada com $3f + 1$ réplicas pode ficar indisponível durante as recuperações quando mais do que f réplicas estão comprometidas ou paradas.

Considere agora que tem uma máquina de estados replicada com n réplicas, capaz de tolerar um máximo de f faltas arbitrárias, e onde os rejuvenescimentos ocorrem em grupos com um máximo de k réplicas. Em qualquer momento, o número mínimo de réplicas garantidamente disponível é dado por $n - f - k$. Logo, em qualquer operação, quer intra-réplicas (por exemplo, uma execução de um protocolo de difusão atómica), quer originada de um participante externo (por exemplo, um pedido de um cliente), um grupo com $n - f - k$ réplicas será usado para executar a operação. Dado que algumas destas operações afectam o estado do sistema replicado, é preciso também garantir que quaisquer dois grupos de $n - f - k$ réplicas se intersectam em pelo menos $f + 1$ réplicas (isto é, dado que f réplicas podem ser maliciosas, esta intersecção garante a participação de pelo menos uma réplica correcta). Portanto, é preciso assegurar que $2(n - f - k) - n \geq f + 1$, e esta condição só pode ser satisfeita se $n \geq 3f + 2k + 1$.

6 Conclusões

Este trabalho apresenta duas contribuições distintas. Na primeira parte do trabalho (Secções 2 e 3) é proposto um modelo que leva em conta a evolução de um determinado recurso durante o tempo de vida de um sistema. Foi definido um novo predicado que permite raciocinar formalmente sobre a possibilidade ou impossibilidade de se conseguir segurança-contra-exaustão. Em termos de possibilidade, foi provado que é exequível construir-se um sistema síncrono tolerante a faltas seguro contra exaustão desde que tenha um tempo de vida limitado e que os pressupostos temporais nunca sejam violados. Contudo, e contra as crenças actuais, também foi provado que é impossível construir-se um sistema tolerante a faltas seguro contra exaustão sob o modelo assíncrono.

O impacto teórico destes resultados é independente do tipo de faltas. Isto é, estes síndromas de falha (por exaustão) eram desconhecidos anteriormente e mesmo com faltas acidentais podem

causar a falha inesperada de sistemas distribuídos assíncronos e síncronos (sem tempo de vida limitado). Consequentemente, estes resultados podem alertar outros investigadores e ajudar a conceber melhores sistemas distribuídos. O impacto prático dos mesmos resultados pode ser cada vez maior, na medida em que os sistemas, críticos ou genéricos, estão a tornar-se presas de ataques de piratas informáticos (faltas maliciosas). Isto significa que, com probabilidade crescente, os sistemas com o síndroma de falha descrito neste trabalho (por exemplo, modelo assíncrono + tempo de exaustão limitado) não apenas podem, como serão mesmo atacados e feitos falhar.

Na segunda parte do trabalho (Secções 4 e 5), foi descrita a resiliência proactiva, um novo paradigma para a construção de sistemas seguros contra exaustão baseado num modelo e arquitectura híbridos: os mecanismos de recuperação proactiva são executados por um subsistema com "melhores" propriedades do que o resto do sistema.

Em termos teóricos, provou-se que a resiliência proactiva permite a construção de sistemas seguros contra exaustão e derivou-se as condições exactas em que tal é possível. Em termos práticos, derivou-se uma máquina de estados replicada segura contra exaustão de nós. Ainda no contexto do cenário da máquina de estados replicada, foi feito um estudo sobre o nível de redundância necessário para se conseguir um sistema resiliente e permanentemente disponível, isto é, garantindo simultaneamente segurança-contra-exaustão de nós e disponibilidade, estabelecendo-se um novo resultado: é necessário um mínimo de $3f + 2k + 1$ réplicas para tolerar f faltas arbitrárias e k recuperações simultâneas.

Referências

[ADD+06] Yair Amir, Claudiu Danilov, Danny Dolev, Jonathan Kirsch, John Lane, Cristina Nita-Rotaru, Josh Olsen, and David Zage. Scaling Byzantine fault-tolerant replication to wide area networks. In *Proceedings of the 2006 International Conference on Dependable Systems and Networks*, pages 105–114. IEEE Computer Society, June 2006.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 164–177. ACM Press, 2003.

[BT85] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, October 1985.

[CKLS02] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *CCS '02: Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97. ACM Press, 2002.

[CKS00] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Contanstinople: Practical asynchronous Byzantine agreement using cryptography. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, pages 123–132. ACM Press, July 2000.

[CL02] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, November 2002.

[CR93] R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 42–51. ACM Press, 1993.

[DGGS99] A. Doudou, B. Garbinato, R. Guerraoui, and A. Schiper. Muteness failure detectors: Specification and implementation. In *EDCC-3: Proceedings of the Third European Dependable Computing Conference on Dependable Computing*, pages 71–87, 1999.

[DSU04] Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421, 2004.

[FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[GGJR00] J. A. Garay, R. Gennaro, C. Jutla, and T. Rabin. Secure distributed storage and retrieval. *Theoretical Computer Science*, 243(1-2):363–389, 2000.

[HJJ+97] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 100–110. ACM Press, 1997.

[HJKY95] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, pages 339–352. Springer-Verlag, 1995.

[HT94] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Cornell University, Department of Computer Science, May 1994.

[Ken80] S. Kent. *Protecting Externally Supplied Software in Small Computers*. PhD thesis, Laboratory of Computer Science, Massachusetts Institute of Technology, 1980.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Lam98] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[Lyn96] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[MR97a] D. Malkhi and M. Reiter. Byzantine quorum systems. In *Proceedings of the 29th ACM Symposium in Theory of Computing*, pages 569–578. ACM Press, May 1997.

[MR97b] D. Malkhi and M. Reiter. Unreliable intrusion detection in distributed computations. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 116–124, June 1997.

[MR00] D. Malkhi and M. Reiter. An architecture for survivable coordination in large distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):187–202, 2000.

[MS04] M. A. Marsh and F. B. Schneider. CODEX: A robust and secure secret distribution system. *IEEE Transactions on Dependable and Secure Computing*, 1(1):34–47, January–March 2004.

[OL88] Brian M. Oki and Barbara H. Liskov. Viewstamped replication: a new primary copy method to support highly-available distributed systems. In *PODC '88: Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing*, pages 8–17. ACM Press, 1988.

[OY91] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 51–59. ACM Press, 1991.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[Rei95] M. K. Reiter. The Rampart toolkit for building high-integrity services. In *Theory and Practice in Distributed Systems*, volume 938 of *LNCS*, pages 99–110. Springer, 1995.

[Sch90] F. B. Schneider. Implementing faul-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[Ske82] Dale Skeen. A quorum-based commit protocol. In *Berkeley Workshop*, pages 69–80, 1982.

[SS92] D. P. Siewiorek and R. S. Swarz. *Reliable Computer Systems: Design and Evaluation (2nd Edition)*. Digital Press, 1992.

[Tru04] Trusted Computing Group. TCG Specification Architecture Overview, revision 1.2. `https://www.trustedcomputinggroup.org/groups/tpm/`, 2004.

[ZSR05] L. Zhou, F. B. Schneider, and R. Van Renesse. APSS: proactive secret sharing in asynchronous systems. *ACM Transactions on Information and System Security*, 8(3):259–286, 2005.

[ZSvR02] L. Zhou, F. Schneider, and R. van Renesse. COCA: A secure distributed on-line certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368, November 2002.